Étape 12 : Déplacement sensoriel, partie 3.

Jamila Sam & Jean-Cédric Chappelier, 2018

Version : 1.0

But

Mettre en œuvre le déplacement sensoriel.

Description

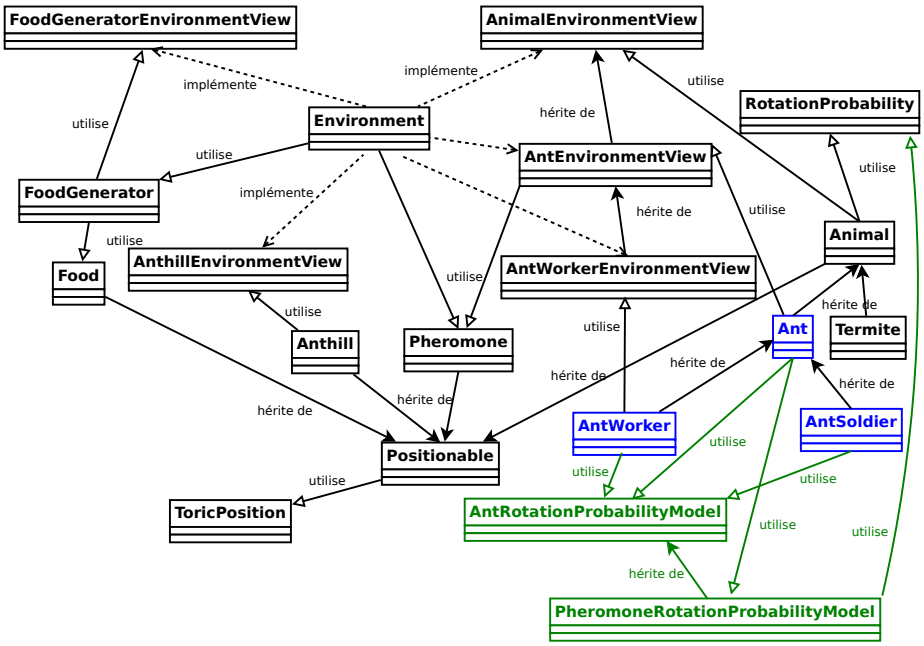

Figure 1: Composants impliqués dans l'étape 12 (en noir les composants déjà codés, en vert les nouveaux composants, en bleu ceux retouchés).

Pour implémenter le déplacement sensoriel, nous allons affiner notre conception

pour permettre l'utilisation flexible de plusieurs modèles de calcul de probabilités de rotation pour les fourmis.

Nous introduirons pour cela l'abstraction `AntRotationProbabilityModel` dont dérivera une nouvelle classe `PheromoneRotationProbabilityModel`. C'est cette classe qui implémentera une stratégie de calcul des probabilités de rotation basée sur la présence de phéromone. Ce calcul sera fait selon un algorithme donné. La figure 1 montre les composants impliqués par cette étape.

Algorithme

Pour mettre en place le déplacement inertiel, nous avions associé deux tableaux au travers du type `RotationProbability` :

- le tableau des angles de rotation,
  par exemple : `I = { -180, -135, -90, -45, 0, 45, 90, 135, 180 }`
- le tableau des probabilités associées à ces angles,
  par exemple : `P = { 0.00, 0.01, 0.09, 0.15, 0.5, 0.15, 0.09, 0.01, 0.00 }`

La simulation du déplacement tenant compte de la présence de phéromone se fait selon un modèle analogue. Ce qui change est pour l'essentiel les modalités de calcul du tableau de probabilité. Nous allons à la place _calcule_r un tableau qui prend en compte la présence de phéromone.

Perception de la phéromone

Soit x une quantité de phéromone proche de la fourmi. Nous allons considérer que le degré de perception de la fourmi (entre 0 et 1) est donné par une fonction de détection suivante (illustrée en figure 2) :

$$D(x) = \frac{1}{1 + \exp(-\beta\,(x - Q_0))}$$

où β est un facteur d'accroissement des capacités de détection sur lequel on peut jouer pour rendre la fourmi plus ou moins sensible à la quantité de phéromone présente (cf fig. 2) et Q_0 est la quantité de phéromone autour de laquelle la détection se fait : la fourmi perçoit pas ou peu les quantités de phéromone inférieures à Q_0 et perçoit (un peu à très bien) les quantités au dessus de Q_0 (cf fig. 2).

En Java, cette formule s'écrit simplement :

```
1.0 / ( 1.0 + Math.exp( -beta * (x - QZero) ) )
```

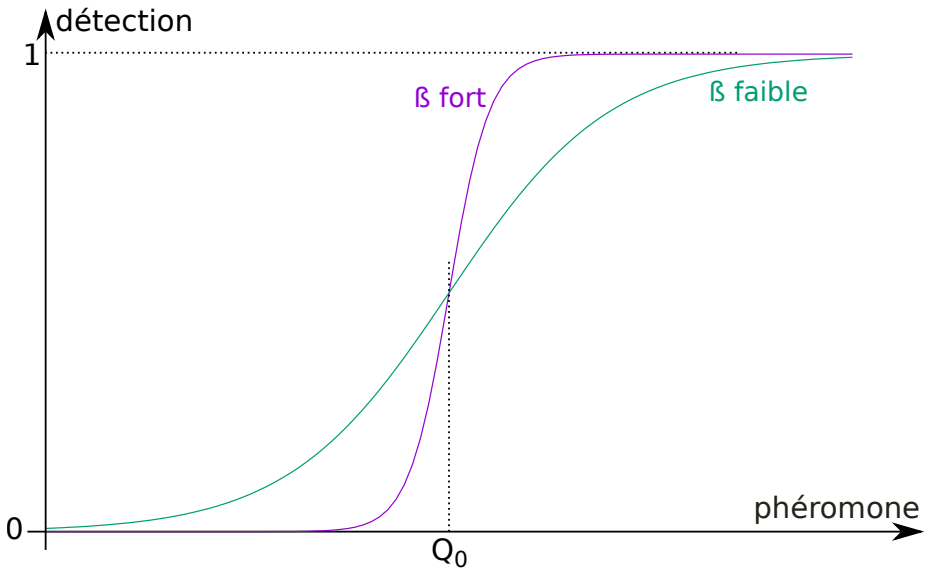

Figure 2: Rôles des paramètres β et Q_0 dans la fonction de détection de phéromone.

Chacun des secteurs définis par le tableau des angles de rotation I peut contenir une certaine quantité de phéromone (à la distance de perception de la fourmi). Le degré de perception de la fourmi pour le secteur correspondant à l'angle `I[i]` est alors simplement donné par l'application de la fonction de perception D précédente à la quantité présente dans ce secteur.

Par exemple, si les quantités présentes dans les secteurs d'angle autour de la fourmi sont données par le tableau

```
Q =  { 20, 0, 0, 2, 0, 0, 10, 0, 5 }
```

(20 unités de phéromone sont derrière à droite de la fourmi, 2 dans le secteur d'angle correspondant à -45 degrés (en fait, entre -67.5 et -22.5 degrés), 10 à gauche (secteur 90 degrés) et 5 derrière à gauche de la fourmi),
alors le degré de perception qu'a la fourmi de ces quantités (pour $\beta = 1$ et $Q_0 = 5$) est donnée par le tableau

```
{ 1, 0.007 , 0.007 ,0.047 , 0.007, 0.007 , 0.993, 0.007 , 0.5 }
```

obtenu en appliquant la fonction D à chacune des entrées du tableau `Q`.

Calcul du nouveau tableau de probabilités de rotation

Le calul du nouveau tableau `P'` des probabilités associées aux angles du tableau `I` se fait ensuite à l'aide des tableaux `P` et `Q` simplement en utilisant la formule

suivante :

$$P'[i] = \frac{P[i] \cdot (D(Q[i]))^{\alpha}}{S}$$

où :

- α est un facteur permettant de « mélanger » la capacité sensorielle des fourmis avec la probabilité inertielle `P` ; il permet de faire augmenter/diminuer l'importance de la détection de phéromone par rapport à la probabilité intertielle ;
- S est simplement la somme de tous les $P[i] \cdot (D(Q[i]))^{\alpha}$.

En Java, cette formule s'écrit simplement :

```
numerateur[i] = P[i] * Math.pow( detection(Q[i]), alpha );
S += numerateur[i];
```

Par exemple, avec les tableaux `P` et `Q` précédents et $\alpha = 5$, on obtient le tableau

```
P' = { 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.00 }
```

les numérateurs respectifs étant :

```
0.00
1.34293736789941e-13
1.20864363110947e-12
3.59888473343695e-08
6.71468683949705e-12
2.01440605184912e-12
0.08702826299281
1.34293736789941e-13
0.00
```

Implémentation

`AntRotationProbabilityModel`

Avant d'attaquer à proprement parler l'implémentation du déplacement sensoriel, nous allons ajouter un niveau d'abstraction supplémentaire, celui de modèle de probabilités de rotation.

Concrètement, nous allons définir une interface `AntRotationProbabilityModel` qui a comme unique méthode `computeRotationProbs`. Cette interface, comme son nom le suggère, représente un modèle de calcul de la matrice des probabilités de rotation. Dans notre cas, le modèle que nous allons utiliser est celui du déplacement sensoriel à l'aide de phéromone, mais si nous souhaitions ajouter

un autre modèle (par exemple un déplacement qui dépend non seulement de la présence de phéromone, mais également de la position du soleil), l'utilisation d'une interface nous permet de faire ce changement avec un minimum de modification dans `Ant`, voir même aucune !

Nous vous proposon de procéder comme suit :

- créer tout d'abord l'interface `AntRotationProbabilityModel` et y déclarer la méthode

  ```
  RotationProbability computeRotationProbs(
                        RotationProbability movementMatrix,
                        ToricPosition position,
                        double directionAngle,
                        AntEnvironmentView env              )
  ```

- créer ensuite la classe `PheromoneRotationProbabilityModel` qui implémente l'interface `AntRotationProbabilityModel` ;. cette classe représente le modèle que nous allons utiliser pour le déplacement sensoriel tenant compte de la présence de phéromone ; pour le moment, `computeRotationProbs` retourne `null` ;

- ajouter le champ (privé !) `probModel` de type `AntRotationProbabilityModel` dans la classe `Ant` ;
  le constructeur actuel de `Ant` initialisera `probModel` à
  `new PheromoneRotationProbabilityModel()` ;
  cela correspond au modèle par défaut que nous allons utiliser ;

- ajouter à `Ant` un constructeur qui prend comme paramètres les mêmes que le constructeur actuel, plus un objet de type `AntRotationProbabilityModel` ; il initialise alors `probModel` avec la valeur passée en paramètre supplémentaire ;

- ajouter un constructeur à `AntWorker` qui a les mêmes paramètres que son constructeur actuel, plus un objet de type `AntRotationProbabilityModel` ; procéder de façon similaire pour `AntSoldier` ;

- enfin, la méthode `computeRotationProbs` de `Ant` se contentera d'appeler la méthode `computeRotationProbs` de `probModel`, avec comme arguments `computeDefaultRotationProbs()`, `getPosition()`, `getDirection()` et `env`.

Comme décrit précédemment, l'ajout d'un niveau d'abstraction supplémentaire comme celui-ci nous permet de changer avec beaucoup plus de flexibilité le modèle de rotation. En effet, si nous souhaitions tenir également en compte d'autres paramètres (par exemple le soleil), il nous suffirait de créer une fourmi avec ce modèle passé en paramètre dans le constructeur. Ainsi, la classe `Ant` ne sera pas touchée par le changement de modèle.

Implémentation du déplacement sensoriel

Il ne nous reste maintenant plus qu'à implémenter la méthode `computeRotationProbs()` de `PheromoneRotationProbabilityModel`. Cette méthode va mettre en œuvre l'algorithme décrit plus haut et calculant le tableau `P'[i]` comme indiqué précédemment :

$$P'[i] = \frac{P[i] * (D(Q[i])^{\alpha}}{S}$$

où :

- Q est l'ensemble des quantités de phéromone perçues telles que calculées par `getPheromoneQuantitiesPerIntervalForAnt` ;
- P est l'ensemble des probabilités de base (telles qu'utilisées pour le déplacement inertiel), correspondant au tableau `I` des angles utilisés par défaut dans la classe `Animal` (paramètre `movementMatrix`).

Les constantes employées dans ce calcul sont :

- le paramètre `ALPHA` (de type `int`) pour α ;
- le paramètre `BETA_D` (de type `double`) pour β ;
- et le paramètre `Q_ZERO` (de type `double`) pour Q_0 .

La méthode `computeRotationProbs()` retournera les tableaux `I` et `P'`.

Note : n'oubliez pas les importations statiques nécessaires à l'utilisation des constantes paramétrables, par exemple:

```
import static ch.epfl.moocprog.config.Config.ALPHA;
```

Tests et soumission

Tests locaux

Nous vous encourageons, comme toujours à d'un coté continuer d'ajouter vos propres tests ponctuels dans le fichier `ch/epfl/moocprog/tests/Main.java` et d'un autre coté utiliser l'interface graphique pour tester globalement le comportement de tout le système.

L'application graphique devrait vous permettre d'observer le déplacement sensoriel : les fourmis qui rencontrent un chemin de phéromone en se déplaçant dévient de leur trajectoire initiale pour suivre le chemin.

Une petite vidéo d'exemple d'exécution est disponible dans la matériel de la semaine 6.

Soumission

Pour soumettre votre devoir au correcteur automatique, il faut créer un fichier ZIP contenant **tous** vos fichiers sources personnels, depuis la racine `ch/` ; ceux de cette étape, mais aussi ceux de l'étape précédente. Concrètement, pour ce devoir ci, votre fichier ZIP doit donc contenir exactement les fichiers suivants :

```
ch/epfl/moocprog/AnimalEnvironmentView.java
ch/epfl/moocprog/Animal.java
ch/epfl/moocprog/AntEnvironmentView.java
ch/epfl/moocprog/AnthillEnvironmentView.java
ch/epfl/moocprog/Anthill.java
ch/epfl/moocprog/Ant.java
ch/epfl/moocprog/AntRotationProbabilityModel.java
ch/epfl/moocprog/AntSoldier.java
ch/epfl/moocprog/AntWorkerEnvironmentView.java
ch/epfl/moocprog/AntWorker.java
ch/epfl/moocprog/Environment.java
ch/epfl/moocprog/FoodGeneratorEnvironmentView.java
ch/epfl/moocprog/FoodGenerator.java
ch/epfl/moocprog/Food.java
ch/epfl/moocprog/Pheromone.java
ch/epfl/moocprog/PheromoneRotationProbabilityModel.java
ch/epfl/moocprog/Positionable.java
ch/epfl/moocprog/RotationProbability.java
ch/epfl/moocprog/Termite.java
ch/epfl/moocprog/ToricPosition.java
```

Note : si c'est plus pratique pour vous, vous *pouvez* aussi faire un zip qui contient tout le sous-répertoire `ch/epfl/moocprog`, y compris, donc, les fichiers que nous vous avons fournis. Ces fichiers seront simplement ignorés par le correcteur automatique (et remplacés par les nôtres). L'énoncé « *Procédure de soumission* » de la semaine 1 vous indique comment procéder. **Avant de passer à l'étape suivante, n'oubliez pas de faire une sauvegarde de l'étape en cours**, comme indiqué dans le même document.